Control

Only 12 classes to go!
John Backus dead at 82
Phase-change Flash memory
Today Control
Next-time Quiz review
John Backus

3 December 1924 – 17 March 2007

As projectleader with IBM John Backus developed in the early 1950's with his team: Fortran - Formula Translator. The first high level programming language. This language is most widely used in physics and engineering.
Phase-Change Flash Memory
Synchronous Systems

On the leading edge of the clock, the input of a latch is transferred to the output and held.

We must be sure the combinational logic has *settled* before the next leading clock edge.
Asynchronous Systems

No clock!
The data carries a “valid” signal along with it
System goes at greatest possible speed.
Only “computes” when necessary.

Everything we look at will be synchronous
Fetching Sequential Instructions

How about branch?
Datapath for R-type Instructions

- Inst Bits 25-21
  - 5
  - Read Reg. 1

- Inst Bits 20-16
  - 5
  - Read Reg. 2

- Inst Bits 15-11
  - 5
  - Write Reg.

- ALU Operation
  - 3
  - data 1
  - 32

- Write Data
  - 32

- RegWrite

- data 2
  - 32
Fun with MUXes

Remember the MUX?

This will route 1 of 4 different 1 bit values to the output.
The select signal determines which of the inputs is connected to the output.
Inside there is a 32 way MUX per bit

For EACH bit in the 32 bit register

LOT’S OF CONNECTIONS!

And this is just one port!
Our Register File has 3 ports

This is one reason we have only a small number of registers

What’s another reason?

REALLY LOTS OF CONNECTIONS!
Implementing Logical Functions

Suppose we want to map M input bits to N output bits

For example, we need to take the OPCODE field from the instruction and determine what OPERATION to send to the ALU.

```
\text{OPCODE bits from instruction} \rightarrow \text{Map to ALU op} \rightarrow \text{ALU Operation}
```
We can get 1 bit out with a MUX

Wire these to HIGH or LOW depending on the value you want OUT for that INPUT.

For example, 3 input AND has INPUT7 wired HIGH and all the others wired LOW.
Or use a ROM

M-bit Address → Read-Only Memory → N-bit Result
Or use a PLA

Programmable Logic Array

Think of the SUM of PRODUCTS form.
The AND Array generates the products of various input bits
The OR Array combines the products into various outputs
Finite State Machines

- A set of STATES
- A set of INPUTS
- A set of OUTPUTS
- A function to map the STATE and the INPUT into the next STATE and an OUTPUT

Remember “Shoots and Ladders”? 
Traffic Light Controller
Implementing a FSM
Recognizing Numbers

Recognize the regular expression for floating point numbers

\[ \text{[ } \text{\tt]* } [-+]?\text{[0-9]*}(. \text{[0-9]*})? (e[-+]?\text{[0-9]+})? \]

Examples:

- \text{+123.456e23} \quad \text{“a” matches itself}
- \text{.456} \quad \text{“[abc]” matches one of a, b, or c}
- \text{1.5e-10} \quad \text{“[a-z]” matches one of a, b, c, d, ..., x, y, or z}
- \text{-123} \quad \text{“0*” matches zero or more 0’s (“”, “0”, “00”, “0000”)}
- \text{Z?” matches zero or 1 Z’s}
### FSM Table

<table>
<thead>
<tr>
<th>IN : STATE</th>
<th>NEW STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(')</td>
<td>start</td>
</tr>
<tr>
<td>(0 \mid 1 \mid \ldots \mid 9)</td>
<td>start whole</td>
</tr>
<tr>
<td>(+\mid-)</td>
<td>start sign</td>
</tr>
<tr>
<td>(\cdot)</td>
<td>start frac</td>
</tr>
<tr>
<td>(0 \mid 1 \mid \ldots \mid 9)</td>
<td>sign whole</td>
</tr>
<tr>
<td>(\cdot)</td>
<td>sign frac</td>
</tr>
<tr>
<td>(0 \mid 1 \mid \ldots \mid 9)</td>
<td>whole whole</td>
</tr>
<tr>
<td>(\cdot)</td>
<td>whole frac</td>
</tr>
<tr>
<td>(\cdot)</td>
<td>whole done</td>
</tr>
<tr>
<td>(\cdot)</td>
<td>whole exp</td>
</tr>
<tr>
<td>(\cdot)</td>
<td>exp done</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STATE ASSIGNMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>start  =  0  =  000</td>
</tr>
<tr>
<td>sign   =  1  =  001</td>
</tr>
<tr>
<td>whole  =  2  =  010</td>
</tr>
<tr>
<td>frac   =  3  =  011</td>
</tr>
<tr>
<td>exp    =  4  =  100</td>
</tr>
<tr>
<td>done   =  5  =  101</td>
</tr>
<tr>
<td>error  =  6  =  110</td>
</tr>
</tbody>
</table>
Our PLA has:

- 11 inputs
- 5 outputs
FSM Take Home

With just a register and some logic, we can implement complicated sequential functions like recognizing a FP number.

This is useful in its own right for compilers, input routines, etc.

The reason we’re looking at it here is to see how designers implement the complicated sequences of events required to implement instructions.

Think of the OP-CODE as playing the role of the input character in the recognizer. The character AND the state determine the next state (and action).